REMARKS

Claims 1-19 have been rejected under 35 U.S.C. §101 for claiming non-statutory subject matter.

Claim 1 has been rejected under 35 U.S.C. §112, second paragraph as being indefinite for failing to particularly point out and distinctly claim the subject matter which the applicant regards as the invention.

Claims 1-19 have been rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 6,230,316 to Nachenberg ("Nachenberg").

Claim 14 has been canceled.

Claims 1-13 and 15-19 remain pending.

Rejection of Claims 1-19 under 35 U.S.C. §101

With respect to claims 1 and 9, the Office Action states that the claims recite a method for enabling a software acquiring entity to exchange two pieces of code, and that this is an abstract idea because the code is not stored. The Office Action further states that the method as recited can be done by a person as a mental step or using a pencil and paper.

Claims 1 and 9 have been amended to add the limitations that the method is computer-implemented method and that the first and second signed pieces of code are executable on a machine. Further, the claim states that the both pieces of code have been generated by first software archive generator. Therefore, the method as recited cannot be accomplished by a person as a mental process or using a pencil and paper.

With respect to claims 13 and 15-19, the Office Action states that the claims recite a code amendment enabler, which is a program product without any hardware to enable the functionality to be realized by causing a computer to perform steps, which would provide a useful, concrete, and tangible result. The Office Action further states that the claims as recited are not necessarily machine implemented and, therefore, are not provided in a manner that would enable its functionality to perform steps that would provide a useful, concrete, and tangible result.

Claim 13 does not recite a code amendment enabler, as indicated in the Office Action. However, claim 13 has been amended to properly recite a program storage device.

Claims 15-19 have been amended to add the limitation that the code amendment enabler is tangibly embodied in hardware.

With respect to claim 14, the Office Action states that the computer readable medium is not limited to tangible embodiments and is therefore not limited to statutory subject matter.

Claim 14 has been canceled.

Applicants respectfully submit that these rejections have now been overcome and that the rejections under 35 U.S.C. §101 be withdrawn.

Rejection of Claim 1 under 35 U.S.C. §112, second paragraph

The Office Action alleges that on line 1 of claim 1 the recited method does not state which steps are necessary to carry out the method. The Office Action further states that it is unclear what steps are considered necessary to arrive from the first piece of code to the second piece of code.

Independent claim 1 has been amended to point out more clearly the steps that are considered necessary to arrive from the first piece of code to the second piece of code.

Applicants submit that this rejection has now been overcome.

Rejection of Claims 1-18 under 35 U.S.C. §102(e)

With respect to independent claims 1, 9, 15, and 18, the Office Action states that Nachenberg discloses all of Applicant's recited elements.

Nachenberg discloses incremental updating of an executable file that has been rebased or realigned. Nachenberg teaches that when an incremental software update is distributed, it may be rebased, which means memory addresses that appear in code and data segments are changed to accommodate the file being loaded into memory at a new base address. Furthermore, the software update may be realigned, which refers to moving the code and data segments within a file. See col. 1, line 66 to col. 2, line 10. Nachenberg states that such rebasing and realigning can result in unpredictable results when a software update occurs by binary patching. Binary patching is a technique for incrementally updating software in which the bits of the software that are different in a new version of the software are changed. See col. 1, lines 39-48, and lines 60-65.

Nachenberg avoids the unpredictable results that may occur due to rebasing or realigning by converting application files to a canonical form before performing an update. For example, as shown in Fig. 8, a version 100A of an application file is converted into a canonical version 100B. An update builder 122 uses a conventional binary patch file builder to determine a binary difference, in an update file 124C-B, between the version 100B file and the new version 100C file. The update file 124C-B is then distributed to a user who installs it on the user's computer

126, which also includes the canonical converter 120. A pre-update version of the file 100 on the user's computer 126 has been rebased and realigned, such that it is in state 100D. The canonical converter 120 on the user's computer 126 converts the file 100D to the canonical version 100B. The update file 124C-B is then used by an updater 128 to produce the new version 100C of the file. The updater 128 can be any conventional binary patcher that applies the patch, e.g., update file 124C-B, to the canonical version 100B. See col. 5, lines 35-60. Thus, the method disclosed by Nachenberg requires that an additional, intermediate file (canonical version) on both the server end and the user end be created before an update is applied.

Nachenberg provides a conventional binary patch file based on the difference between an updated file and a canonical version of a pre-update file. At the user's location, the binary patch file is applied to a canonical version of the pre-update file to derive the updated file.

In contrast, Applicants' invention involves providing information to an end user to allow the user to update a first code to a second code. This is achieved by providing generation instructions and a difference code to the user. The generation instructions are instructions that are used by a first software archive generator to generate both the first and second code at the location of a software provider. The difference code includes the steps necessary to arrive at the second code from the first code. The user employs a second software archive generator, which uses the difference code and the first code to generate the second code based on the generation instructions. Applicant's invention does not require creating an intermediate, canonical version of the code before an update is applied.

Further, Nachenberg does not teach or suggest the use of a second software archive generator (on the user's side) that is fed with generation instructions that were used by a first software archive generator (on the server side) for generating both the first and second pieces of

code. There is no teaching in Nachenberg that the update file 100C is generated from the pre-update file 100A using generation instructions. Instead, it is only assumed that the update file 100C is somehow available. Nachenberg is concerned with is providing a file that represents a binary difference between the file 100C and the canonical version 100B of file 100A. Nachenberg also teaches that the updater 128 uses the update file 124C-B to produce version 100C. However, the updater 128 is any conventional binary patcher, which can apply patches. The updater 128 is clearly not the same as Applicants' second software archive generator, which uses the generation instructions that were used by a first software archive generator to generate both first and second pieces of code.

Further, in the Office Action, under the section "Response to Arguments", the Examiner failed to respond to Applicants' argument that Nachenberg also does not teach or suggest that the first and second pieces of code are both signed pieces of code.

In view of the foregoing, it is respectfully submitted that Nachenberg, does not teach or suggest the subject matter recited in Applicants' independent claim 1 as this reference fails to teach or suggest a computer-implemented method for a software provider of enabling a software-acquiring entity to arrive from an existent first signed piece of code executable on a machine at a second signed piece of code executable on the machine, where both pieces of code have been generated by use of a first software archive generator under use of generation instructions. The method includes providing to the software-acquiring entity a difference code, where the difference code includes the steps necessary to arrive from the first signed piece of code at the second signed piece of code, where the difference code is usable at the software-acquiring entity. The method further includes combining the difference code with the first signed piece of code by a second software archive generator to generate the second signed piece of code, whereby the

second software archive generator is fed with those generation instructions that were used by the first software archive generator for the generation of both pieces of code.

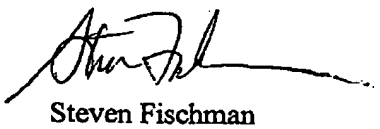
Independent claims 9, 15, and 18 recite similar features as claim 1, and therefore are patentably distinct over Nachenberg for at least the reasons discussed in connection with independent claim 1.

Claims 2-8, 10-13, 16-17, and 19 which depend directly from the independent claims 1, 9, 15, and 18, incorporate all of the limitations the corresponding independent claims and are therefore patentably distinct over Nachenberg for at least those reasons provided for claims 1, 9, 15, and 18.

Conclusion

In view of the foregoing, Applicants respectfully request reconsideration, withdrawal of all rejections, and allowance of all pending claims in due course.

Respectfully submitted,



Steven Fischman
Registration No. 34,594

Scully, Scott, Murphy & Presser
400 Garden City Plaza, Suite 300
Garden City, N.Y. 11530
(516) 742-4343

SF/BMM:ej